一种基于二进制编码的 Apriori 改进算法 *

胡世昌，李劲华[†]，王常颖

(青岛大学，数据科学与软件工程学院，山东 青岛 266071)

摘 要：Apriori 算法在挖掘频繁项集时需要多次扫描数据库，这样会因为频繁的 IO 操作而导致效率低下。为了改进算法的执行效率，BE-Apriori(binay encoded Apriori)算法充分利用了二进制数相比编程语言中各种数据结构在内存及运算速度上的优势，对事务记录进行二进制编码后加载到内存，然后利用等效的二进制数之间运算代替集合之间的运算。分析了算法性能，并利用 UCI 数据集中的毒蘑菇数据对 BE-apriori 算法进行实验验证。结果表明 BE-Apriori 可以正确挖掘频繁项集，并且相比 Apriori 算法有着更好的性能。

关键词：频繁项集；集合运算；二进制；Apriori 算法

中图分类号：TP301.6 doi: 10.19734/j.issn.1001-3695.2018.07.0519

Improved Apriori algorithm based on binary encoding

Hu Shichang, Li Jinhua[†], Wang Changying

(*School of Data Science & Software Engineering, Qingdao University, Qingdao Shandong* 266071, *China*)

Abstract: Apriori algorithm has to scan the database multiple times when mining frequent item sets, resulting in inefficiencies because of frequent IO operations. To improve efficiency of Apriori algorithm, BE-Apriori algorithm makes full use of the advantages of binary numbers compared to the memory usage and computational speed of various data structures in programming languages. It loads the transaction record with binary encoding into memory through, then translates the set operations into the equivalent binary number operations. The analysis of algorithm performance and the experimental result using the poisonous mushroom data in the UCI data show that BE-Apriori can correctly mine frequent itemsets and has better performance than the original Apriori algorithm.

Key words: frequent itemsets; set operations; binary; Apriori

0 引言

随着社会经济和互联网的快速发展，信息资源彻底打破之前地域和时空的限制，在网络上飞速地传播和发展。信息存储的单位从之前的以 G 为单位，到现在的以 PB 甚至 EB 为单位。然而，随着信息量的增加，无效信息也越来越多，从海量信息中挖掘特定信息的难度也越来越大。因而从大量信息中挖掘有效信息的技术越来越重要，数据挖掘技术和算法应运而生，关联规则挖掘是其中的一个重要研究方向，有着广泛的应用。

Apriori 算法是关联规则挖掘算法中最为经典的一种，它是由美国学者 Agrawal[1]在 1993 年提出。Apriori 算法在各种领域应用良好，能够有效地分析海量事务之间的相关性，在决策制定中提供有效的支持[2]。Apriori 算法主要由两步组成，从事务记录中获取频繁项集、根据频繁项集获取关联规则[3]。对 Apriori 算法的性能起决定性作用的是第一步[4]，获取所有的频繁项集。之后的基于 Apriori 算法的改进的算法主要也是对获取频繁项集的改进。一种基于矩阵的 Apriori 改进算法[5]，将事务记录转换为矩阵的形式，减少了遍历数据库的次数，然后通过对矩阵的运算来获取频繁项集。然而矩阵的运算时间较长。基于向量矩阵优化频繁项的改进 Apriori 算法[6]，通过运用快速排序的思想对频繁项集的项按各单项的出现频度升序重排，以此来减少候选频繁项集的产来提高算法的执行效率。该算法性能不稳定，通过排序来减少候选项集的产生，但是不能过滤掉不可能的全部候选项集。一种 Apriori 的改进算法[7]通过使用概率的方法估算数据项集同时出现的概率，但其中的参数 a 和 b 的设定需要额外的时间，而且还存在着频繁项集的缺失的可能性。基于十字链表的 Apriori 算法[8]提出一种方法，将事务记录映射到十字链表，有效地组织了事务记录排列，可以减少访问数据库的次数以及扫描事务记录的次数。基于预判筛选的高效关联规则挖掘算法[11]首先随机采样统计频繁项集然后再计算原始数据的频繁项集，引入阻尼因子和补偿因子对预判筛选产生的误差进行修正，可以在一定误判率和遗漏率的情况下提高 Apriori 算法的执行效率。这些算法对 Apriori 算法上提出了改进，但是结果却并不是很理想。为了进一步的提高算法的执行效率，提出了基于二进制编码的 BE-Apriori 算法。

1 Apriori 算法

Apriori 算法通过通过层层迭代的方式逐层获取频繁项集。通过性质 1 和 2 减少候选项集的个数，避免了产生太多的候选项集从而过多地扫描数据库计算项集的支持度。

1.1 性质及定义

性质 1 频繁项集的任何非空项集都是频繁项集。

性质 2 非频繁项集的超集是非频繁项集。

支持度是指一个项集的支持度是包含该项集的事务在事

收稿日期：2018-07-11；**修回日期：**2018-08-22 **基金项目：**全国统计科学研究项目（2017LY14）

作者简介：胡世昌（1993-），男，湖北广水人，硕士研究生，主要研究方向为机器学习；李劲华（1963-），男（通信作者），湖南长沙人，教授，博士，主要研究方向为软件工程、算法理论、大数据理论与技术（lijh@qdu.edu.cn）；王常颖（1980-），女，副教授，主要研究方向为海洋复杂性与数据挖掘.

物记录中的比例。其中"包含"的意思是指项集是事务的子集。

1.2　算法描述

本文就寻找频繁项集这一步上来提高 Apriori 算法的效率。Apriori 算法中挖掘频繁项集的步骤如下：

a)扫描数据库，获取所有项的支持度，获取频繁 1-项集。

b) 连接。若两个频繁 k 项集 A :{A[1],A[2],,,A[k-1],A[k]},B:{B[1],B[2],,,B[k-1],B[k]}满足如下条件:A[1]=B[1],A[2]=B[2],,,A[k-1]=B[k-1],A[k]!=B[k]，则项集 A 和项集 B 可以进行连接构成候选频繁(k+1)-项集，{A[1],A[2],,,,A[k],B[k]}。

c)剪枝。根据 Apriori 算法的性质 2，非频繁项集的超集也是非频繁项集，对产生的候选项集进行剪枝。若 k 个候选频繁(k+1)-项集的每个 k-项项集不是都是频繁 k-项集，即有一个或者多个 k-项子集不是频繁 k-项集，那么该候选频繁(k+1)-项集不可能为频繁(k+1)-项集，以此来对候选项集进行剪枝。

d)通过遍历数据库，对候选频繁(k+1)-项集的支持度进行统计，若低于最小支持度，就进行滤除。在进行支持度统计的时候对于每个候选(k+1)-项集，对其中的每个项都需要遍历一次数据库，即需要遍历(k+1)次才可以统计出该频繁(k+1)-项集的支持度[6]。

e)循环执行步骤 b)~d)，直到不能通过连接产生候选频繁项集为止。

1.3　Apriori 算法的缺陷

Apriori 算法的缺点主要如下(通过频繁 k-项集产生频繁(k+1)-项集来举例)：

a)连接和剪枝的步骤计算效率低下，在判断了两个频繁 k-项集中有(k-1)项是相同的基础上，需要对产生的候选项(k+1)-项集的 k 个 k-项集都是频繁 k-项集做判断。

b)每个候选(k+1)-项集是否是频繁 k-项集还需要遍历(k+1)次数据库，但是 IO 效率是非常低下的。

c)在扫描数据库时需要对候选项集和事务进行模式匹配，花费大量的时间[9]。

2　BE-Apriori 算法

众所周知，I/O 存取的消耗相对于内存存取要高几个数量级[10]。本文提出的 BE-Apriori 算法，在 Apriori 算法的三个步骤上分别都提出改进，提高了挖掘频繁项集的效率。本文算法只需要遍历两遍数据库，第一遍，统计得到频繁 1-项集；第二遍，根据频繁 1-项集对事务记录进 2 编码，存入内存；之后的全部计算都可以在内存中计算，可以有效的避免由于 IO 效率的低下而造成的时间损耗。

2.1　基本概念：

若频繁 1-项集的个数为 n，可以把所有项集编码为长度小于或等于 n 位的二进制数。长度为 n 的二进制数中的每个位置分别代表每个项。若某项在此项集中存在，则在该位置为 1；若该项在此项集中不存在，则该位置为 0。编码后的二进制数可能小于 n，这是由于二进制数前面部分位置代表的项均不存在。据此可以对所有事务，频繁项集和候选项集进行编码。然后，通过长度小于或等于 n 的二进制数代表它们。

性质 3　连接频繁 k-项集 A 和 B 得到候选(k+1)-项集 C，若候选项集 C 中的项集{A[k],B[k]}不是频繁项集，则候选项集 C 不可能是频繁项集。

证明　由 Apriori 算法的性质 2 可以知道，非频繁项集的超集也是非频繁项集。由于候选项集 C 的子集{A[k],B[k]}不是频 繁项集，所以候选项集 C 不可能是频繁项集。

性质 4　编码后的项集 a，和项集 b。a&b 的结果为项集 a 和项集 b 共同含有的项集;例如若频繁 1-项集为{A,B,C}，项集 a-{A，B}二进制编码的结果为 110，项集 b-{A,B,C}二进制编码的结果为 111，项集 c-{A,C}二进制编码的结果为 101。a&b=a，表明{A，B}为{A，B，C}的子集，a&c！=a，表明{A,B}并非{A,C}的子集。

性质 5　编码后的项集 a，和项集 b。a^b 的结果为项集 a 和项集 b 的不同的项集。例如若频繁 1-项集为{A,B,C}，项集 a-{A，B}二进制编码的结果为 110，项集 b-{A,B,C}二进制编码的结果为 111，a^b=001，表明项集 a 和项集 b 中不同时拥有的项集为 001，即为{C}。

2.2　实例介绍

现举例说明 BE-Apriori 算法，事务记录如表 1 所示，设定的最小支持度为 0.4。

表 1　事务记录

Table 1　Transaction record

TID	list of items
T1	A,B,C,D
T2	C,E
T3	C,D
T4	A,C,D
T5	C,D,E

算法运行步骤如下：

a)扫描数据库，获取各个 1-项集的支持度，滤除支持度小于最小支持度的候选 1-项集，统计得到的频繁 1-项集及其编码结果为表 2 所示。

表 2　频繁 1-项集及其编码结果

Table 2　Frequent 1-item set and its encoded result

频繁 1-项集	编码结果
A	1000
C	100
D	10
E	1

b)根据频繁 1-项集对事务记录进行编码，由于频繁 1-项集的个数为 4，事务记录的编码长度小于或等于 4。可以得到事务记录的编码结果如表 3 所示。

表 3　事务记录的编码结果

Table 3　Encoded transaction record

TID	编码结果
T1	1110
T2	101
T3	110
T4	1110
T5	111

c)通过遍历频繁 1-项集产生候选 2-项集，由于产生候选 2-项集的时候，不能进行剪枝操作，所以产生的候选 2-项集的规模会比较大。产生的候选 2-项集及其编码结果如表 4 所示。

d)计算每个候选 2-项集的支持度。候选 2-项集 1100 与各事务与的结果如表 5 所示。

根据性质 4 可以统计得到候选 2-项集 1100 与各事务记录与的结果等于候选 2-项集自身的个数为 2，于是可以得到 1100 的支持度为 0.4。

同理可得，1010 支持度为 0.4，1001 支持度为 0，0110

支持度为 0.8，0101 的支持度为 0.4，0011 支持度为 0.2。于是频繁 2-项集 $L_2=\{1100,1010,0110,0101\}$

表 4　候选 2-项集及其编码结果

Table 4　Candidate 2-item set and its encoded result

候选 2-项集	编码结果
A,C	1100
A,D	1010
A,E	1001
C,D	110
C,E	101

表 5　候选 2-项集 1100 与事务记录与的结果

Table 5　Result of AND operation between candidate frequent 2-items set 1100 and encoded transaction record

TID	与事务记录与的结果
1110	1100
0101	100
0110	100
1110	1100
0111	100

e)获取候选 3-项集。根据性质 5，由于 1100^1010=1100，表明频繁 2-项集 1100 和频繁 2-项集 1010 不同时含有的项集为 1100，由于 1100 属于频繁 2-项集，所有 1100+1100&1010 为候选 3 项集。同理可得，再无其他候选项集。于是，候选 3-项集为 1110。

f)获取频繁 3-项集。1110 与各事务与的结果如表 6 所示。

表 6　候选三项集 1110 与事务记录与的结果

Table 6　Result of AND operation between candidate frequent 3-items set 1110 and encoded transaction record

事务记录	与事务记录与的结果
1110	1110
0101	100
0110	110
1110	1110
0111	110

所以 1110 的支持度为 0.4，得到频繁 3-项集为 1110。由于只有一个，故不能连接生成候选 4-项集。到此结束。

2.3　BE-Apriori 算法描述

设 L_k 表示频繁 k-项集，C_k 表示候选频繁 k-项集，D 表示事务记录

BE-Apriori 算法的伪代码如下：

输入：事务记录 D，支持度 minsup

输出：频繁项集 L

forall transactions $t\in D$ *do begin*
　forall $ele\in t$ *do begin*
　　ele.count++
　end
end
$L_1=\{ele|ele.count\geq minsup\}$
for (k=2; $L_{k-1}\neq\varnothing$; k++) *do begin*
　if k==2 *then*
　　C_k =apriori-gen-1(L_{k-1});
　else
　　C_k =apriori-gen-2(L_{k-1});
　　forall transactions $t\in D$ *do begin*
　　　forall $c\in C_k$ *do begin*
　　　　if t & c == c *then*
　　　　　c.count++
　　　　end
　　　end
　　end
$L_k=\{c\in C_k|c.count\geq minsup\}$
Answer= $U_k L_k$
//获取候选 2-项集
apriori-gen-1
Insert into C_k
select p.item$_1$, q.item$_1$
from L_{k-1} p, L_{k-1} q
//获取候选 k-项集(k>2)
apriori-gen-2
insert into C_k
select p.item$_1$, p.item$_2$,,, p.item$_{k-1}$, q.item$_{k-1}$
from L_{k-1} p, L_{k-1} q
where p^q $\in L_2$

2.4　BE-Apriori 算法性能分析

a)避免了多次扫描数据库，仅仅扫描两遍数据库，就可以通过编码后的事务集代替数据库中的事务集。之后仅仅通过在内存中的运算，就可以得到全部频繁项集。可以有效的减少 Apriori 算法中由于频繁扫描数据库进行的 I/O 操作所耗费的时间。

在 Apriori 算法中的连接操作需要判断两个频繁 k-项集的前(k-1)项是否相同，这个需要在项集按照特定顺序排列的情况下才能比较， k 项集的排序在较好的情况下的时间复杂度为 $O(k\log k)$ 。在 Apriori 算法中的剪枝操作，需要判断候选(k+1)-项集的 k 个 k-项子集，这个操作需要的时间复杂度为 $O(k)$ 。所以连接和剪枝操作的时间复杂度为 $O(k^2\log k)$ 。而在 BE-Apriori 算法中，连接和剪枝的操作在 1 步完成，通过编码后的两个频繁 k-项集异或的结果是否为频繁 2-项集，就可以完成连接和剪枝的操作，其时间复杂度为 $O(1)$ ，即通过常数时间就可以完成。

b) Apriori 算法性质 1 之后，绝大部分运算都可以归结为集合之间的运算。通过二进制编码后的项集，只需要通过计算机底层所支持的二进制的基本计算就可以代替集合之间的运算。集合在编程语言中需要转换为特定的数据结构，再进行集合之间的运算。而二进制编码后的项集可以省略中间步骤，从而可以有效的提高 BE-Apriori 算法的执行效率。

3　实验验证

实验环境：处理器为 2.7 GHz Intel Core i5，内存是 8 GB 1867 MHz DDR3，操作系统为 macOS 10.13.5，选用了 Python3.6 作为开发语言，分别实现了基本的 Apriori 算法，VM_Apriori 算法[6]和 BE-Apriori 算法。实验数据采用的数据集为 Frequent Itemset Mining Dataset Repository 中的蘑菇数据。实验目的是挖掘与蘑菇毒属性相关联的属性。数据集中的事务数为 8124，事务的长度均为 23，总共包含 120 个项。表 7 给出了在支持度为 0.2，三个算法在事务记录分别为 1000,2000,4000,8000 的时候，事务记录分别占用的内存。图 1 给出了支持度分别设置为 0.12，0.15，0.17，0.2，两个算法各自的运行时间。

由表 7 可知，在相同事务记录的情况下，BE-Apriori 算法中经过二进制编码后的事务记录占用的内存是远小于

Apriori 算法中原始的存储方式和 VM_Apriori 算法中矩阵的存储方式。由此可见，BE-Apriori 算法为把事务记录加载到内存中提供了可行性。

表 7　三个算法分别占用内存 /Byte

Table 7　三个算法分别占用内存 /Byte

算法事务数	Apriori	VM_Apriori	BE-Apriori
1000	2 274 272	1 135 298	28 028
2000	4 546 272	3 021 596	56 028
4000	9 090 272	5 911 368	112 028

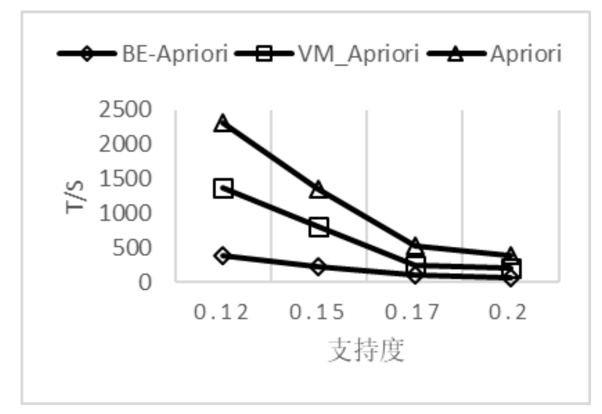


图 1　三个算法的运行时间

Fig. 1　Runtime of three algorithms

在支持度较小的时候，频繁 1-项集会较多，由于算法是层层迭代的，所以会造成之后的计算量也会增多，也就是算法的运行时间会随着支持度的增大而减少。从图 1 中可以明显的看出，BE-Apriori 算法在支持度较小的时候，其运行时间都是远小于另外两种算法；在支持度较大的时候，耗费时间也是小于两外两种算法的。

实验结果表明，该算法在时间和空间上的效率相较于 Apriori 算法是有显著提升的。

4　结束语

从 Apriori 算法到 BE-Apriori 算法，没有复杂的推导，简单易于理解。综合考虑了 Apriori 算法挖掘频繁项集过程中的缺陷，并提出相应的解决方案。创造性的提出以二进制编码的项集作为载体载入内存，并在二进制编码的基础上有效的进行等效的集合之间的运算。通过实验对比可知，该算法有效的提高了 Apriori 算法的执行效率和空间利用率。然而，在频繁 1-项集较大的情况下，项集的编码长度较长。这种情况下，项集占用的内存会较大。以后的工作可以在编码方式上展开。

参考文献：

[1] Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large database [C]// Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 1993: 207-216.

[2] 雷学锋. 基于关联规则的矿井监控数据挖掘分析 [J]. 煤炭技术, 2017(11): 289-291. (Lei Xuefeng. Mining data analysis of mine monitoring based on association rules [J]. Coal Technology, 2017(11): 289-291.)

[3] 曲睿，张天娇. 基于矩阵压缩的 Apriori 改进算法[J]. 计算机工程与设计，2017,38(8): 2127-2131. (Qu Rui, Zhang Tianqiao. Improved Apriori algorithm based on matrix compression [J] , Computer Engineering and Design, 2017,38(8): 2127-2131.)

[4] 涂明，张公让，程业媛. 一种高效的关联规则增量式更新算法 [J]. 微电子学与计算机，2010(9): 56-60. (Tu Ming, Zhang Gongrang, Cheng Yeyuan. An efficient incremental Updating Algorithm for mining association rules [J]. Microelectronics & Computer, 2010(9): 56-60.)

[5] 王蒙，邹书蓉，方睿. 一种基于矩阵的 Apriori 改进算法 [J]. 信息技术，2018(3): 150-154, 158. (Qu Meng, Zou Shurong, Fang Rui. An improved Apriori algorithm based on matrix [J]. Information Technology, 2018(3): 150-154, 158.)

[6] 曹莹，苗志刚. 基于向量矩阵优化频繁项的改进 Apriori 算法[J]. 吉林大学学报：理学版, 2016,54(2): 349-353. (Cao Ying, Miao Zhigang. Improved Apriori algorithm based on vector matrix optimization frequent items [J]. Journal of Jilin University: Science Edition, 2016, 54(2): 349-353.)

[7] 陈江平，傅仲良，徐志红. 一种 Apriori 的改进算法 [J]. 武汉大学学报：信息科学版, 2003,28(1): 94-99. (Cheng Jiangping, Fu Zhongliang, Xu Zhihong. An improved algorithm of Apriori [J]. Geomatics and Information Science of Wuhan University. 2003,28(1): 94-99.)

[8] 黄建明，赵文静，王星星. 基于十字链表的 Apriori 改进算法 [J]. 计算机工程, 2009,35(2): 37-38, 41. (Huang JianMing, Zhao Wenjing, Wang Xingxing. improved Apriori algorithm based on across linker [J]. Computer Engineering, 2009,35(2): 37-38, 41.)

[9] 崔贯勋，李梁，王柯柯，等. 关联规则挖掘中 Apriori 算法的研究与改进 [J]. 计算机应用，2010, 30(11): 2952-2955. (Cui Guanxun, Li Liang, Wang Keke, Gou Guanglei, *et al.* Research and improvement on Apriori algorithm of association rule mining [J]. Journal of Computer Applications, 2010,30(11): 2952-2955)

[10] Ramalho L. 流畅的 Python [M]. 安道,吴珂,译。北京：人民邮电出版社, 2017: 797-798. (Luciano Ramalho. Fluent Python [M]. An Dao,Wu Ke translated. Beijing: Posts & Telecon Press, 2017: 797-798.)

[11] 李德辰，吕一帆，赵学健. 一种基于预判筛选的频繁项集挖掘算法 [J]. 计算机技术与发展，2018,28(5): 99-102. (Li Dechen, Lyu Yifan, Zhao Xuejian. A frequent item-set mining algorithm based on prejudgment and screening [J]. Computer Technology and Development, 2018,28(5): 99-102.)